Express Mail Label # EK830786335US

DE9-2000-0027US1

Europäisches Patentamt	European Patent Office	Office européen des brevets

Bescheinigung | Certificate | Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr.	Patent application No.	Demande de brevet n°
	00106869.1	

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

I.L.C. HATTEN-HECKMAN

DEN HAAG, DEN
THE HAGUE, 12/10/00
LA HAYE, LE

Europäisches Patentamt | European Patent Office | Office européen des brevets

Blatt 2 der Bescheinigung
Sheet 2 of the certificate
Page 2 de l'attestation

Anmeldung Nr.: Application no.: Demande n°: 00106869.1

Anmeldetag: Date of filing: Date de dépôt: 30/03/00

Anmelder: Applicant(s): Demandeur(s):
International Business Machines Corporation
Armonk, NY 10504
UNITED STATES OF AMERICA

Bezeichnung der Erfindung: Title of the invention: Titre de l'invention:
System and method for realizing transactions supported by a directory access protocol

In Anspruch genommene Priorität(en) / Priority(ies) claimed / Priorité(s) revendiquée(s)

Staat: State: Pays:	Tag: Date: Date:	Aktenzeichen: File no. Numéro de dépôt:

Internationale Patentklassifikation: International Patent classification: Classification internationale des brevets:
/

Am Anmeldetag benannte Vertragstaaten: Contracting states designated at date of filing: AT/BE/CH/CY/DE/DK/ES/FI/FR/GB/GR/IE/IT/LI/LU/MC/NL/PT/SE Etats contractants désignés lors du depôt:

Bemerkungen: Remarks: Remarques:

DESCRIPTION

System and method for realizing transactions supported by a directory access protocol

The present invention relates to a system and method for realizing transactions by using controls of a protocol. More specifically, the present invention relates to system and method to secure consistency of related entries in a Directory Information Tree (DIT) when entries are updated or extended by using a Directory Access Protocol especially LDAP V3 (Lightweight Directory Access Protocol).
In the information technology protocols define set of rules that regulate the way data is transmitted between computers.

LDAP V3 is a protocol designed for accessing information stored in a DIT (Directory Information Tree). LDAP V3 is a proposed Internet standard and is disclosed in IETF RFC #2251. LDAP V3 enables LDAP clients to send requests for searching and / or updating entries in the DIT to an LDAP server. The LDAP server performs the client's request and sends back a response containing a return code.

LDAP V3 only allows atomic requests. A single request belonging to a sequence of requests is performed independently of the others. In particular, if one request of the sequence fails, the LDAP Server does not rollback the effects of the requests that were already successfully performed.

LDAP V3 allows so called controls: LDAP requests can be extended with additional information, i.e. a control name and a control value. The control name is a unique LDAP object identifier (OID). The LDAP server has to decide how to handle a request containing controls. The LDAP client can specify that the LDAP server must recognize a control (control is critical) or is requested to recognize a control (non critical). Also the LDAP server can add controls to the responses it sends to LDAP clients.

Assuming two entries A and B in the DIT are strongly related in the way that entry B has to be updated whenever entry A is updated. If an LDAP client sends two subsequent update requests RA (on entry A) and RB (on entry B) to the LDAP server and RA succeeds and RB fails, the effects of RA have to be rollbacked to keep the DIT consistent.

The typical solution for this problem known in computer technology is so called transactions: A whole sequence of requests is performed as if it were a single atomic request. Either the whole sequence of requests is performed successfully or none of the requests will be performed. A transaction can be opened explicitly, closed explicitly (commit), or caused to fail explicitly (rollback).

LDAP V3 does not contain the transaction concept. So far, LDAP clients cannot run transactions against an LDAP server since LDAP V3 does not contain syntactical means to work with transactions.

Patent abstract of Japan JP 11096062 discloses a directory access method for securing consistency in directory information. The method is based on a directory server and a client which are connected by a network. A data base for storing and managing directory information is connected to the directory server. The directory server has a non-transaction processing part for processing respective access requests as different action. A transaction processing part processes series of access requests as a single transaction. A phase managing table stores a processing phase for each connection with the client. Based on the stored processing phase, a phase managing part delivers the accepted access request to the non-transaction part or to the transaction processing part. The disclosure does not teach how the inventive directory access method may be used within an existing protocol without adapting the protocol itself.

It is therefore object of the present invention to provide a system and method for realizing transactions by using rules

preferred embodiment of the present invention if the client wants to commit an open transaction directly after having performed a request the request has to be extended with control TxnControl and the value COMMIT. If the client wants to rollback an open transaction without having performed the request the request has to be extended with the control TxnControl and the value ROLLBACK. The inventive method and system is preferably used within LDAP using V3 controls.

The present invention is described in detail using a preferred embodiment with figures, where

FIG 1A shows a preferred embodiment of a client-server architecture in which the present invention may be used

FIG 1B shows another embodiment of a client-server architectur in which the present invention may be used

FIG 1C-D shows a structure of a Directory Information Tree used by the LDAP protocol

FIG 2 A-B shows basic methods for realizing transaction based on LDAP according to the present invention

FIG 3 A-M shows different response behavior of the LDAP server based on the information contained in the requests.

FIG 1A shows a typical client-server architecture used by the present invention. On the client system, an application and an LDAP client program are installed. The application communicates with the LDAP client, e.g. generates and sends update requests (add, modify, delete a entry in a DIT) to the LDAP client program.
The LDAP client program communicates with a LDAP server program via network.

LDAP client as well LDAP server uses for example the LDAP protocol based on TCP/IP. LDAP client program adds a TransactionControl to a request for opening a transaction as taught by the present invention and sends that request to the LDAP server program. The LDAP server program generates a TransactionID and returns a return code to the LDAP client program containing at least that Transaction ID. The LDAP client program extracts TxnId from the return code and adds a TransactionID to each single request belonging to that transaction. In a preferred embodiment LDAP server program delegates the management of the DIT to the backends. Each Backend 1,2,3 owns a specified DIT portion 1,2,3.

FIG 1 B shows another embodiment of a client-server architecture used by the present invention.

In FIG 1B LDAP Server manages the DIT itself. Requests may be performed by buffering or journalling.

Buffering: LDAP client program sends a request extended by TransactionControl to the LDAP server program. The LDAP server program opens a new transaction by generating a queue in which all requests belonging to the transaction are stored. If the queuing was successful,the LDAP server immediately generates a response which is sent to the LDAP client program.

Journalling: LDAP client program sends a request extended by TransactionControl to the LDAP server program. The LDAP server program opens a new transaction, generates a TransactionId and immediately executes that request. The status of data of the Directory Information Tree before executing that request will be stored on a non-volatile storage media.

FIG.1C shows a DIT as used by LDAP. DIT consists of entries, e.g. countryName, organizationName, organisationUnit, commonName. Each entry has an object class (e.g. person). Object class determines attributes. Attributes have values of a certain type.

FIG 1D shows the structure of an address of entry in a DIT. Address is distinguished name(DN). Address of an entry concatenates all RDNs on path from entry to root. DN suffix denotes sub tree.

FIG 2A shows the basic method for realizing transactions supported by LDAP according to the present invention. In order to perform a transaction containing the sequence of LDAP update requests R1 R2 ... Rn, an LDAP client has to do the following:

Add control TransactionControl with value NEW to request R1 to open a new transaction. The control TransactionId must not be specified.

All subsequent requests R2 ... Rn have to be extended (at least) with the control TransactionId which will be generated by the LDAP server. The value of the control TransactionId must be a valid ID for an open transaction. The LDAP server sends a response containing such an Id when opening a new transaction (see below).

If an LDAP client wants to commit an open transaction directly after having performed Rn, it has to extend Rn with the control TransactionControl and value COMMIT. The control TransactionId also has to be specified with appropriate value.

FIG 2B shows the same transaction method as shown in FIG 2A however with control TransactionControl ROLLBACK.

If an LDAP client wants to rollback an open transaction without having performed Rn, it has to extend Rn with the control TransactionControl and value ROLLBACK. The control TransactionnId also has to be specified with appropriate value.

As laid down above, at least following control information are needed and have to be supported by the LDAP server to realize transactions:

TransactionControl(TxnControl):

OID: A unique LDAP object identifier.
Description: Control used on first and last request of a request sequence that is to be considered as a transaction.
Criticality: Always critical.
Possible Values: Value is exactly one char 0 terminated string in UTF-8 encoding representing exactly one of the strings (words) NEW, COMMIT, and ROLLBACK. Case of these strings is insensitive.

TransactionId(TxnId):

OID: A unique LDAP object identifier.
Description: Indicates the transaction ID assigned to the transaction that the request is a part of.
Criticality: Always critical.
Possible Values: Value is exactly one char 0 terminated string in UTF-8 encoding representing a nonnegative, nonzero long int value (less equal 2,147,483,647 = (2^31)-1) in decimal format which is the transaction id. (Only values formerly received from the LDAP server are allowed - all others are rejected.).

LDAP update requests are Modify, Add, Delete, and ModifyDN.

FIG 3 A - M shows different response behaviour of the LDAP server based on the information contained in the requests.

This is an example for the standard case in LDAP V3: no transaction support is available. All requests are performed in sequential order. The single requests have no relationship from the LDAP server point of view (see FIG 3A).

If no transaction facilities are available from the LDAP server/Backend, the LDAP server has to track whether one single request belonging to a sequence of related requests fails. In the case of request failure, the LDAP client has to build the requests manually which restore the old data (see FIG 3 B).

Request R contains neither TransactionControl nor TransactionId: R is performed as single atomic request. None of the controls TransactionControl and TransactionId are added to the response (see FIG 3 C).

Request R which is no update request and contains either TransactionId or TransactionControl or both: R is not performed and the response has contains result code „unwillingToPerform". None of the controls TransactionControl and TransactionId are added to the response (see FIG 3 D).

Request R contains at least TransactionControl with syntactically invalid value: As specified in RFC 2251. R is not performed. Request R contains at least TransactionId with syntactically invalid value: As specified in RFC 2251. R is not performed (see FIG 3E).

Request R contains at least TransactionId with an Id which does not denote an open transaction: R is not performed and the response contains result code „unwillingToPerform". None of the controls TransactionControl and TransactionId are added to the response (see FIG 3F).

Request R contains TxnId with a valid Id and does not contain TransactionControl:
Perform R. If R can be performed successfully TransactionId with the appropriate Id is added to the response. TransactionControl is not contained in the response. If R fails, all of the effects caused by requests belonging to the transaction identified by Id are rollbacked, the transaction identified by Id is closed, and the response is extended with TransactionId (value Id) and TransactionControl (value ROLLBACK - see FIG 3G).

Request R contains TransactionId with a valid Id and TransactionControl with value „NEW" R is performed. The transaction identified by Id remains open. The response is extended with TransactionnId (value Id) and TransactionControl

(value NEW - see FIG 3H).

Request R contains TransactionId with a valid Id and TransactionControl with value COMMIT:
Perform R. If R can be performed successfully, TransactionId (value Id) and TransactionControl (value COMMIT) are added to the response and the transaction identified by ID is closed (i.e. commited). If R fails all of the effects caused by requests belonging to the transaction identified by Id is closed, and the response is extended with TransactionId (value Id) and TransactionControl (value ROLLBACK - see FIG 3I).

Request R contains TransactionId with a valid Id and TransactionControl with a value ROLLBACK. R is not performed. All of the effects caused by requests belonging to the transaction identified by ID are rollbacked, the transaction identified by Id is closed, and the response is extended with TransactionId (value Id) and TransactionControl (value ROLLBACK- see FIG 3J).

Request does not contain TransactionId and contains TransactionControl with the value NEW: A new transaction with identifier Id is opened. If this operation fails, R is not performed and the response contains the result code „unwilling ToPerform". The response is extended with TransactionControl (value NEW). The control TransactionId is not contained. If a new transaction can be performed successfully opened, R is performed. If R can be performed successfully, TransactionId (value ID) and TransactionControl (value NEW) are added to the response and the transaction identified by Id remains open. If R fails, all of the effects caused by R are rollbacked, the transaction identified by Id is closed, and the response is extended with TransactionControl (value NEW). The control TransactionId is not contained (see FIG 3K).

Request R does contain TransactionId and contains TransactionControl with the value COMMIT: R is not performed and the response contains the result code „unwillingToPerform". None

of the controls TransactionControl and TransactionId are added to the response (see FIG 3L).

Request R does not contain TransactionId and contains TransactionControl with the value ROLLBACK: R is not performed and the response contains the result code „unwillingToPerform". None of the controls TransactionControl and TransactionId are added to the response (see FIG 3 M).

Summarizing, the present invention allows transactional LDAP clients to work with non-transactional LDAP servers if transactions are not used. If, in this case, transactions are used, the LDAP server will deny transactions although it does not know about transactions. This is a major advantage of our invention.

Non-transactional LDAP clients can work with transactional LDAP servers without problems.

Extended requests simply contain a unique object identifier (OID) and a (string) value. (In fact, they contain the same information as an LDAP V3 control.) Extended requests are stand alone requests which do not refer to specific entries in the DIT managed by the LDAP server. LDAP V3 controls, on the other side, are added to "normal" requests like add, delete, or search which always refer to entries or DIT sub parts.

LDAP servers may delegate the management of certain DIT sub parts to so called backends. This is, in fact, what the OS/390 Security Server LDAP Server does: The whole DIT is partitioned into sub trees which are managed by backends. The LDAP server simply chooses the appropriate backend for handling a request. In order to be able to do so, the incoming request must contain a reference to an entry or DIT sub part.

If an incoming request contains no reference to the DIT, the LDAP server cannot delegate the request to a backend since no backend

is appropriate. This is the case for LDAP V3 extended requests. They can be only handled by the LDAP server itself and cannot be delegated without further rules. Such rules might be specified: Extended requests with OID x are handled by backend y. But these rules are very specific and not part of LDAP V3.

That's why the realization of transactions using controls is much easier than using extended requests. Appropriate "TransactionControls" are simply added to requests and will be routed to the appropriate backend, automatically. So, only the backend is responsible for supporting the controls and realizing transactional behaviour. With this solution, the LDAP server does not need to know the OIDs of the controls and does not have to do anything to support transactions.

This is different with an extended request solution: At least, the LDAP server has to know the OIDs of the "transaction extended requests" and has to inform the appropriate backend(s). In the worst case, the LDAP server has to support transactions itself.

C L A I M S

1. Method for realizing transactions with an existing protocol used by different system components communicating with each other, wherein each transaction contains update requests belonging together and the update request is generated by one system component and transmitted to another system component allowing access to a storage media on which information to be updated is stored, comprising:

   a) opening a transaction by automatically adding controls (TransactionControl) to said update request (extended update request) supported by the protocol securing exchange of update requests between said system components

   b) automatically generating a Transaction Identifier(TransactionId) for said transaction supported by said protocol

   c) automatically adding said TransactionId to each update request belonging to said opened transaction

   d) automatically closing said transaction by using a TransactionControl supported by said protocol.

2. Method according to claim 1, wherein said system component for generating said update request is installed on a client system and said system component allowing access to information to be updated is installed on a server system.

3. Method according to claim 2, wherein said TransactionId is generated by said server system.

4. Method according to claim 2, wherein said TransactionId is generated by said client system.

5. Method according to claim 2, wherein said TransactionId is sent from said server system to said client system.

6. Method according to claim 1, wherein step a) is performed with the first update request of a sequence of update requests to be considered as transaction.

7. Method according to claim 1, wherein said update request is a delete, modify or add request.

8. Method according to claim 1, wherein said control for opening a transaction according to step a) has the value NEW.

9. Method according to claim 1, wherein said step d) is performed with the last update request of a sequence of update requests to be considered as transaction.

10. Method according to claim 1, wherein said control for closing a transaction according to step d) has the value COMMIT or ROLLBACK.

11. Method according to claim 1, wherein all extended update requests belonging to a transaction are stored into a queue and performed as a whole when an update request containing control value COMMIT is received.

12. Method according to claim 1, wherein each single extended update request belonging to a transaction is performed immediately after having stored data to be updated in a non-volatile storage media.

13. Method according to claim 1, wherein said protocol is LDAP V3.

14. Method according to claim 13, wherein said controls consists of a control name and a control value, wherein said name is

a unique LDAP object identifier and said value is a binary data.

15. Method according to claim 1, wherein said information to be updated are stored in a Directory Information Tree.

16. Method according to claim 1, wherein an update request containing neither TransactionControl nor TransactionId is performed as single atomic request.

17. Method according to claim 1, wherein an update request with a TransactionControl with syntactically invalid value is not performed.

18. Method according to claim 10, wherein all update requests identified by the same TransactionId are performed and TransactionControl (value COMMIT) and TransactionId are added to the response and the transaction identified by said TransactionId is closed if an update request containing TransactionControl with the value COMMIT and a valid TransactionId is received by said server system.

19. Method according to claim 10, wherein all effects caused by said update requests belonging to said transaction identified by said TransactionId are rollbacked, said transaction is closed and a response is extended with TransactionControl (value ROLLBACK) and said TransactionId if said update request cannot be performed successfully.

20. Method according to claim 8, wherein a new transaction is opened when said update request does not contain Transaction Id and contains TransactionControl with the value NEW.

21. Method according to claim 1 wherein all effects caused by said update requests are rollbacked, the transaction identified by said TransactionId is closed and a response is

extended with a TransactionControl (valueNEW) not containing said TransactionId if the first request of a transaction cannot be performed successfully.

22. System for updating information comprising:

a client system containing at least a client program for adding TransactionControl information and TransactionId to update requests according to the claim 1

a server system containing at least a server program for generating TransactionId according to claim 1 and accessing or updating information

a data storage media containing information to be accessed or updated.

23. System according to claim 22 wherein said client program is a LDAP client program and said server program is a LDAP server program and said information to be updated or accessed are laid down in a Directory Information Tree.

24. System according to claim 23 further comprises:

at least one backend for managing a subtree of said Directory Information Tree

a component for routing said update request to the backend managing information to be updated which is part of said server program.

25. System comprising:

a client data processing system

a client program installed on said system comprises at least:

a component for adding TransactionControl information and TransactionId to update requests according to claim 1.

26. System comprising:

a server data processing system

a server program installed on said system comprises at least:

a component for generating TransactionId according to claim 1

a component for accessing or updating data.

27. Computer program product stored in the internal memory of a digital computer, containing parts of software code to execute the method in accordance with 1 to 21 when said program product is running on said computer.

A B S T R A C T

The present invention discloses a system and method for realizing transactions within existing protocols for accessing information. The present invention uses control information (control) supported by the protocol for specifying transactional or non-transactional requests. This will be achieved by adding controls, e.g TransactionControl and Transaction Identifier (TransactionId), to single requests.

TransactionControl may be a control for opening or closing a transaction. TransactionId identifies each single request belonging to certain transaction. A transaction is opened by a client program by adding TransactionControl with the value NEW to the first request. Preferably, the server program generates a TransactionId for the opened transaction and returns a return code at least containing said TransactionId. All subsequent requests belonging to that transaction have to be extended with the TransactionId by the client program. The transaction will be closed by adding TransactionControl COMMIT or ROLLBACK to the single request. The inventive method and system is preferably used within LDAP using V3 controls. (Fig. 2A)

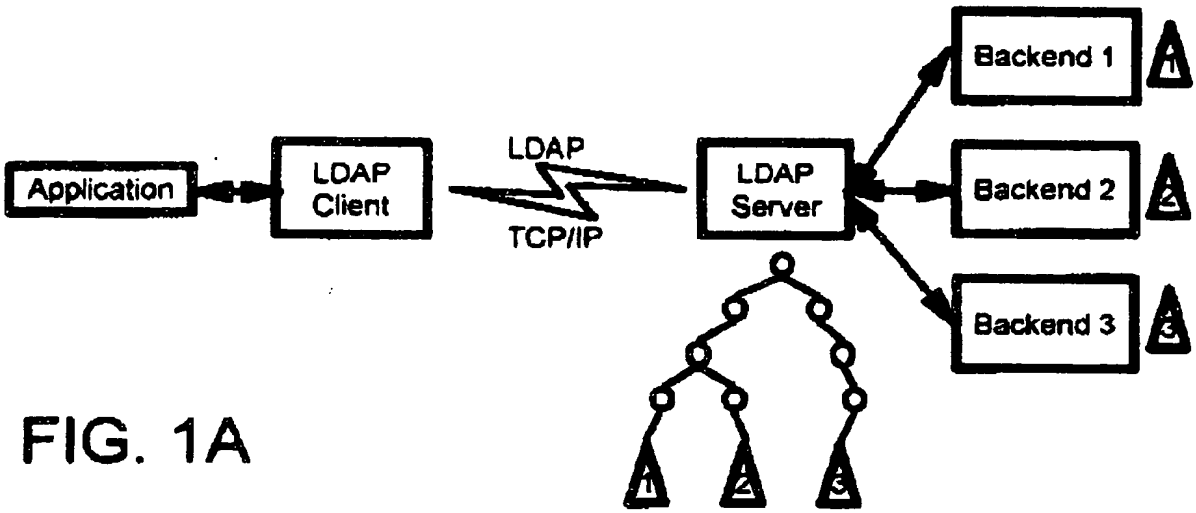

FIG. 1A

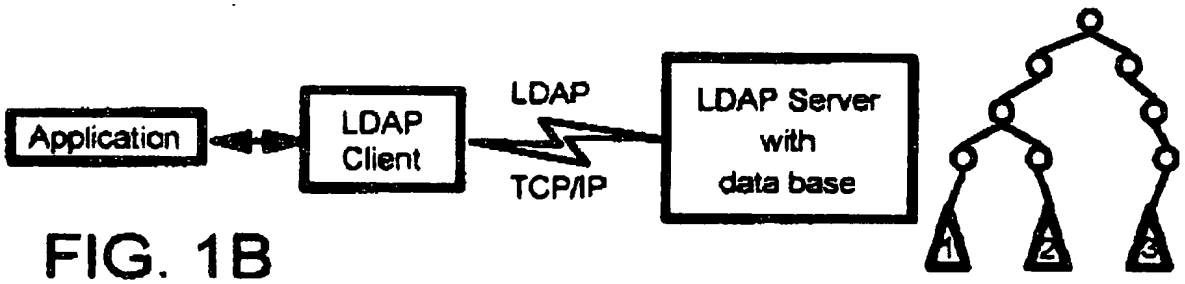

FIG. 1B

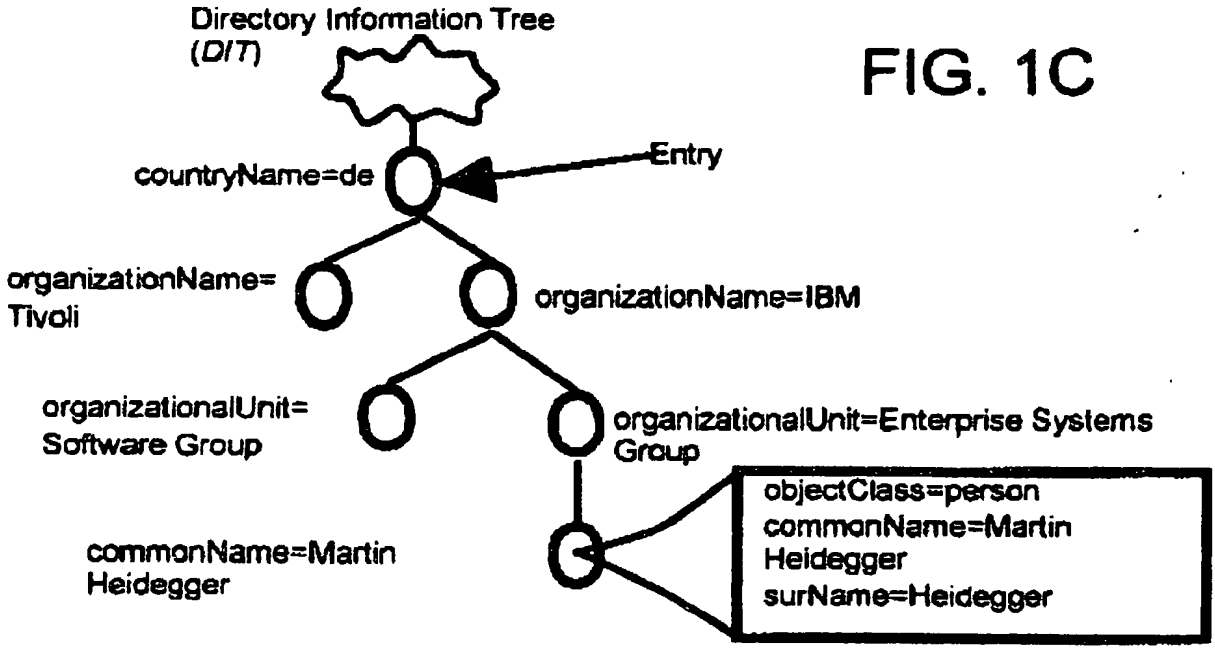

FIG. 1C

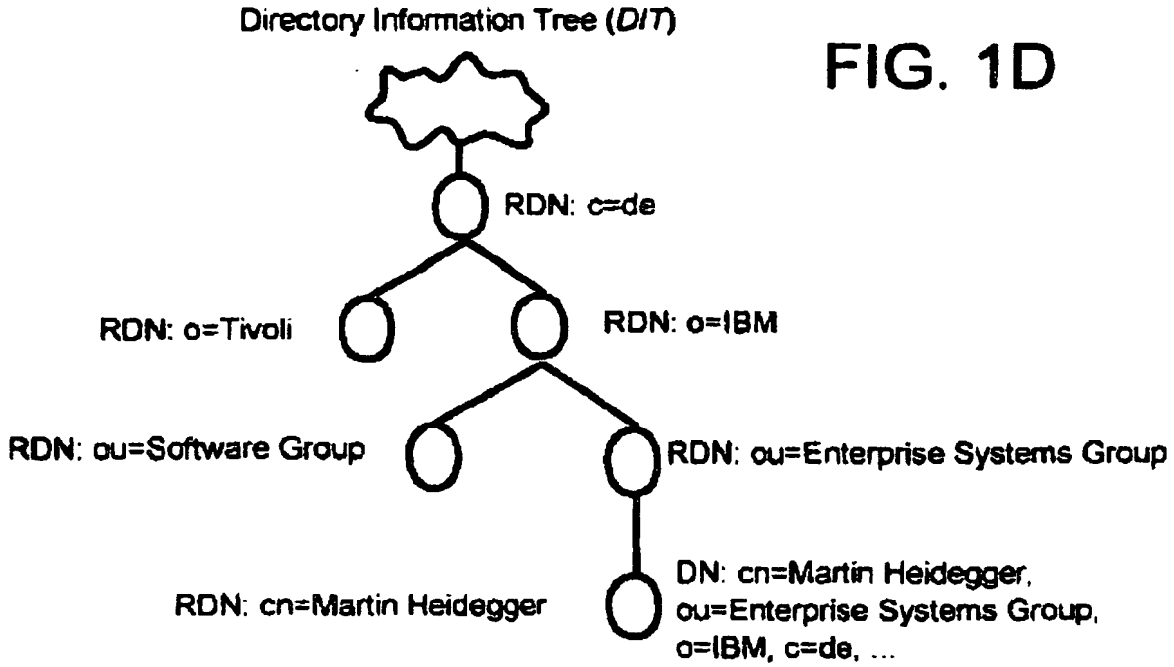
Directory Information Tree (DIT)
RDN: c=de
RDN: o=Tivoli
RDN: o=IBM
RDN: ou=Software Group
RDN: ou=Enterprise Systems Group
RDN: cn=Martin Heidegger
DN: cn=Martin Heidegger, ou=Enterprise Systems Group, o=IBM, c=de, ...

FIG. 1D

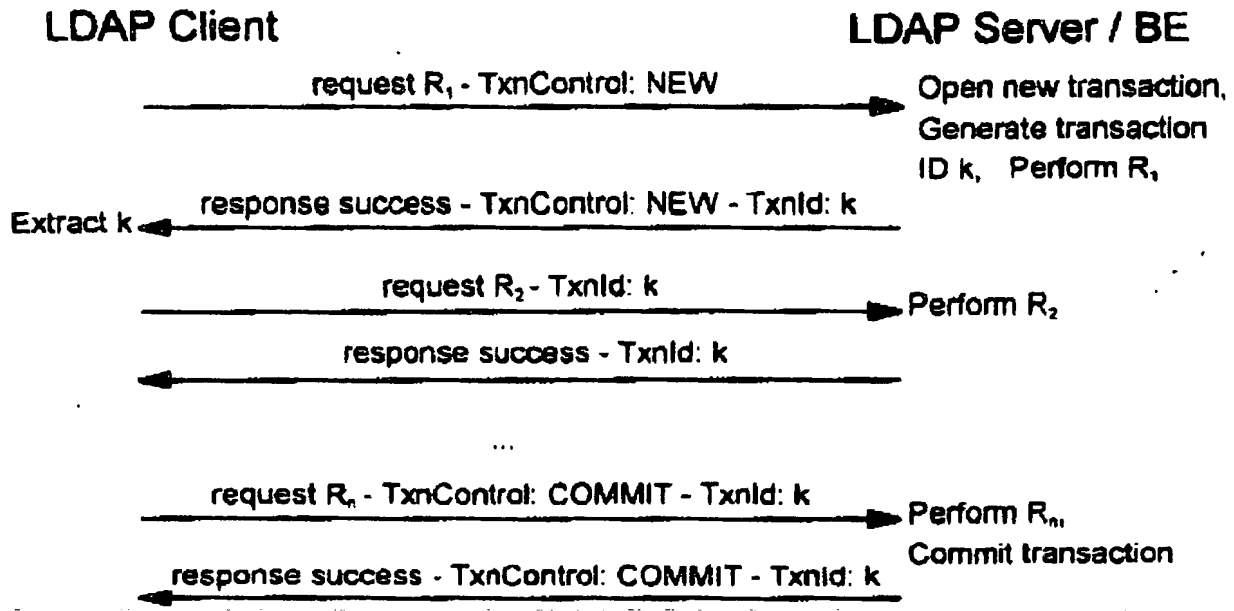
LDAP Client
LDAP Server / BE
request R1 - TxnControl: NEW
Open new transaction, Generate transaction ID k, Perform R1
Extract k
response success - TxnControl: NEW - TxnId: k
request R2 - TxnId: k
Perform R2
response success - TxnId: k
...
request Rn - TxnControl: COMMIT - TxnId: k
Perform Rn, Commit transaction
response success - TxnControl: COMMIT - TxnId: k

FIG. 2A

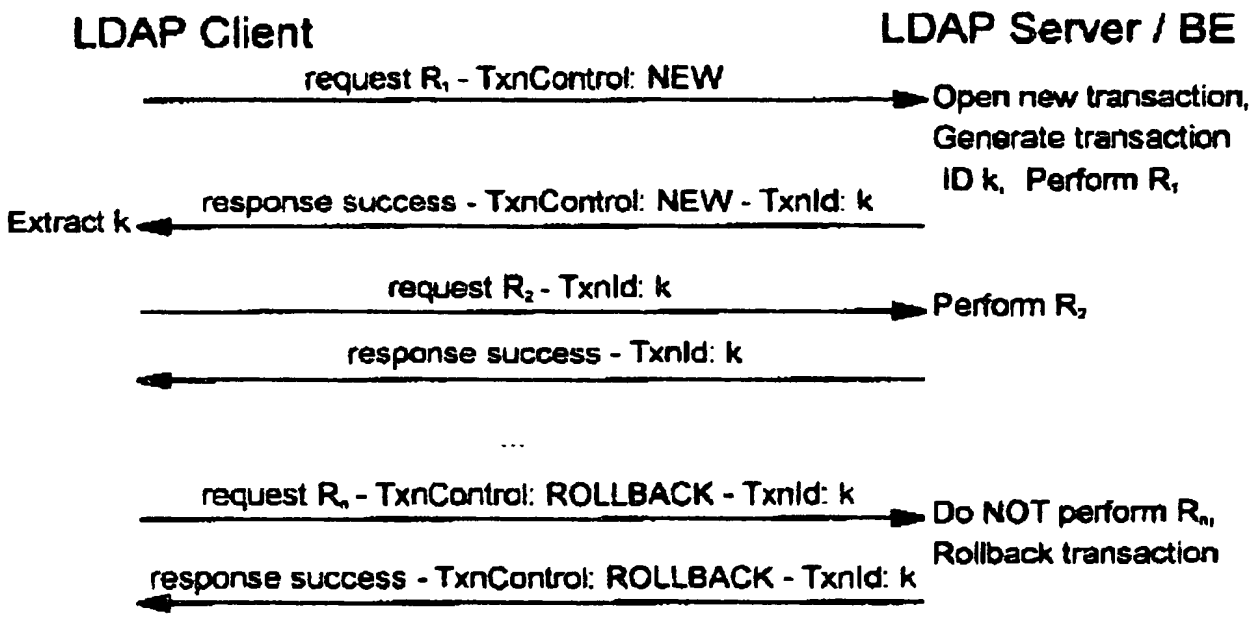
LDAP Client
LDAP Server / BE
request R_1 - TxnControl: NEW
Open new transaction, Generate transaction ID k, Perform R_1
response success - TxnControl: NEW - TxnId: k
Extract k
request R_2 - TxnId: k
Perform R_2
response success - TxnId: k
...
request R_n - TxnControl: ROLLBACK - TxnId: k
Do NOT perform R_n, Rollback transaction
response success - TxnControl: ROLLBACK - TxnId: k

FIG. 2B

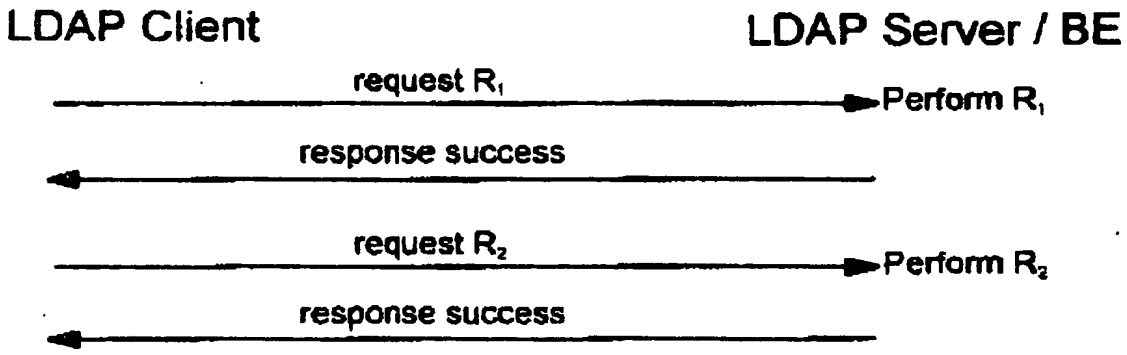
LDAP Client
LDAP Server / BE
request R_1
Perform R_1
response success
request R_2
Perform R_2
response success

FIG. 3A

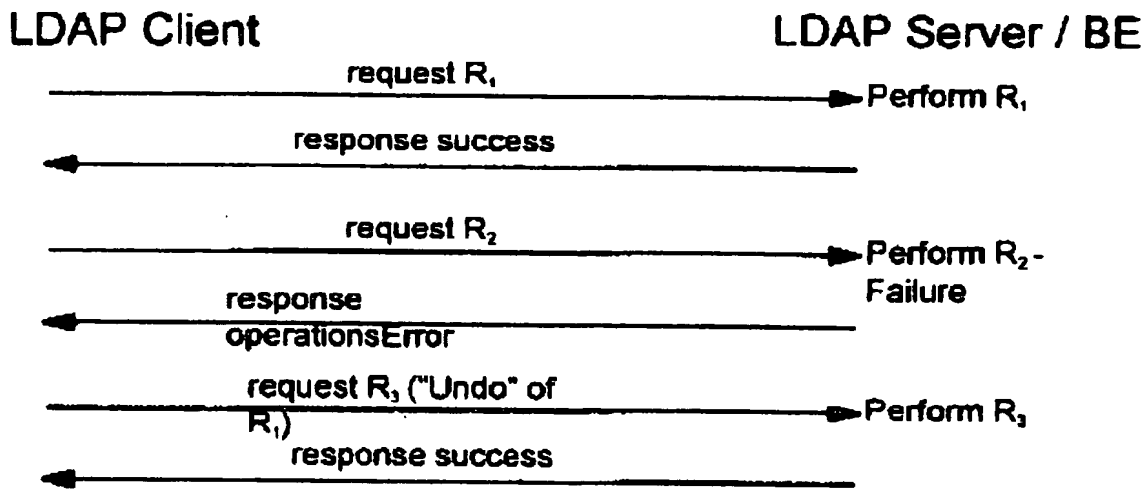

FIG. 3B

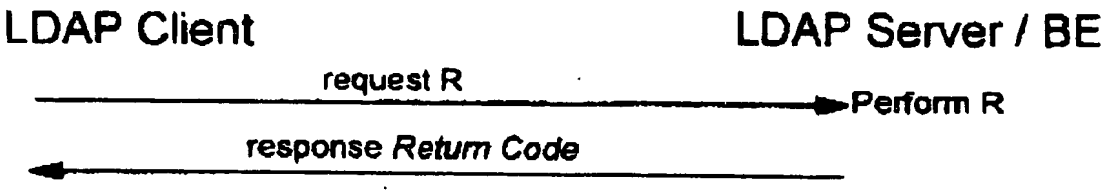

FIG. 3C

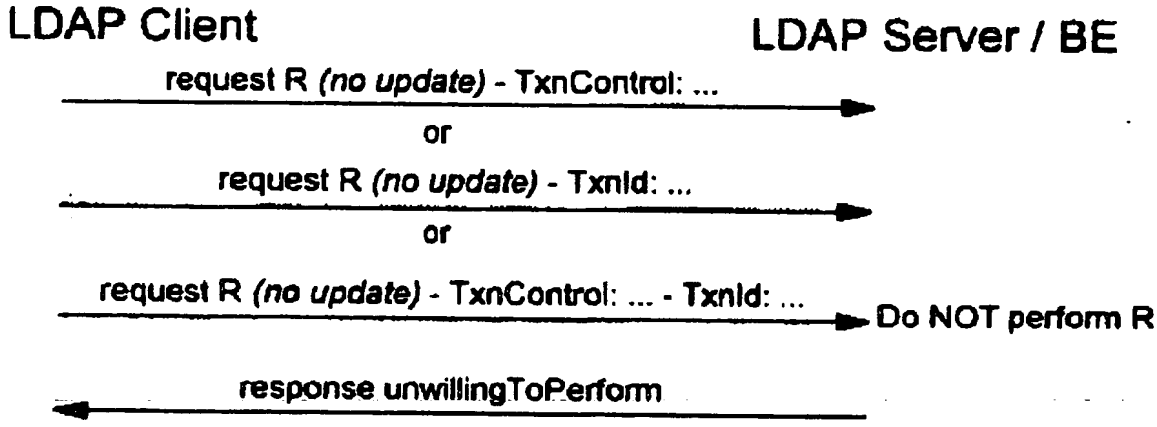

FIG. 3D

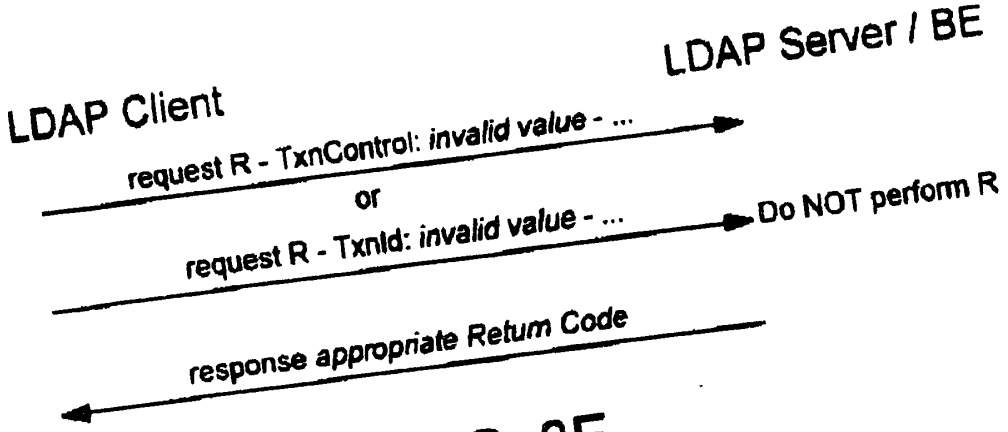

FIG. 3E

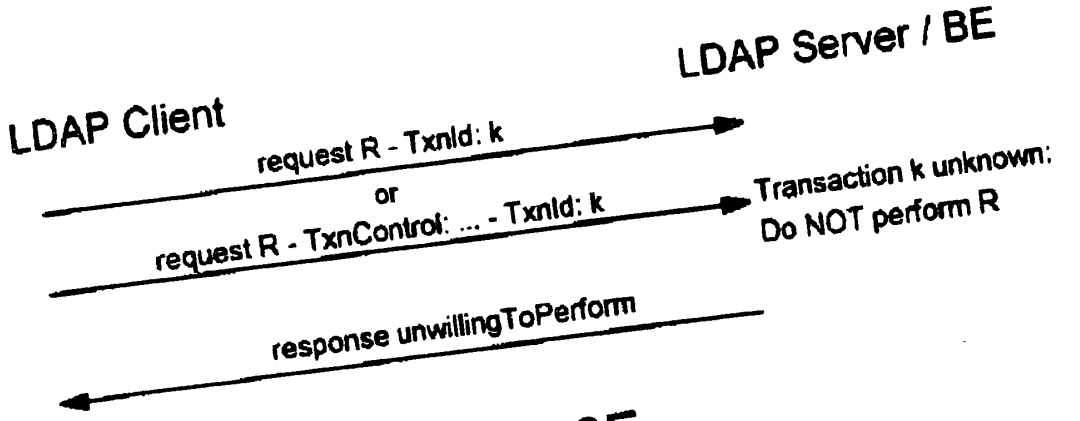

FIG. 3F

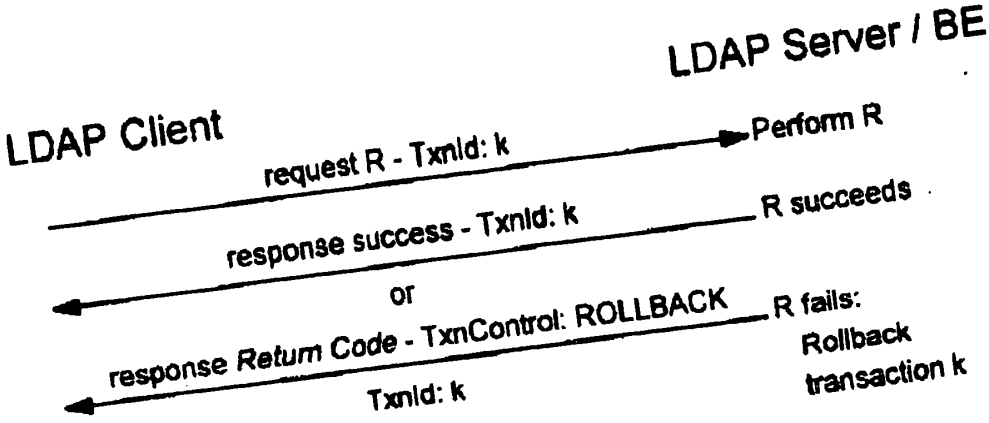

FIG. 3G

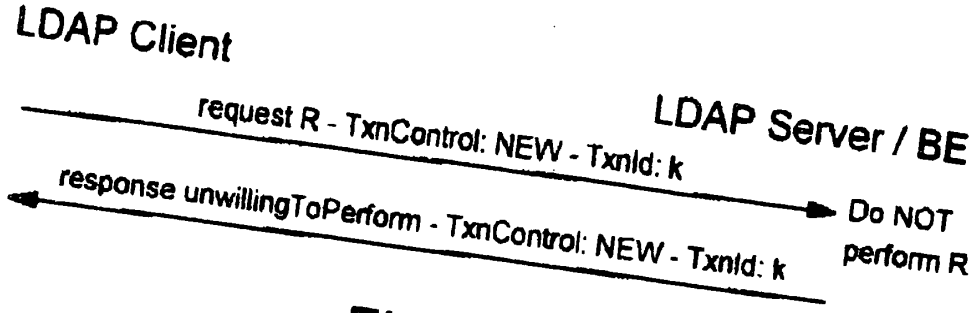

FIG. 3H

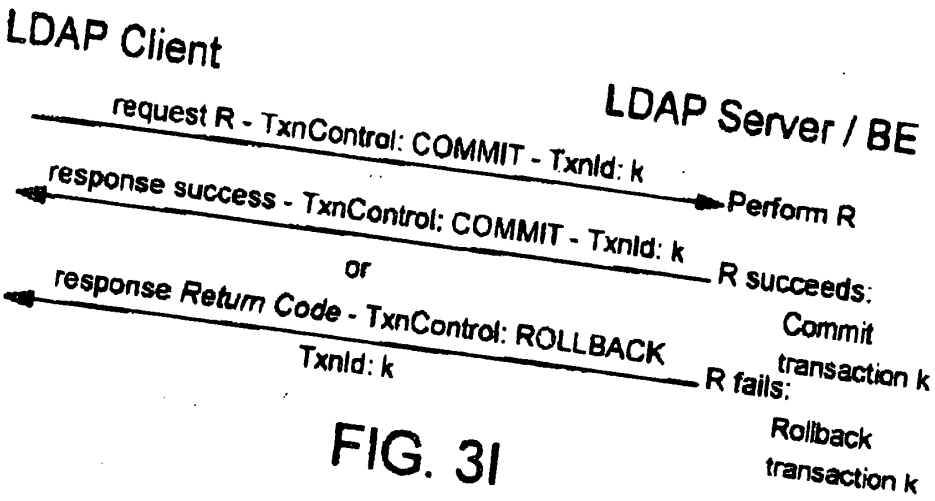

FIG. 3I

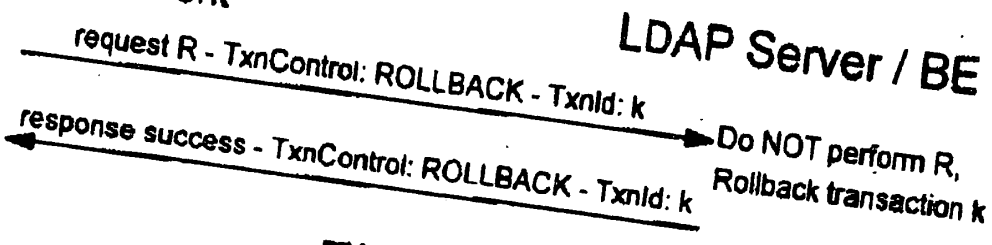

FIG. 3J

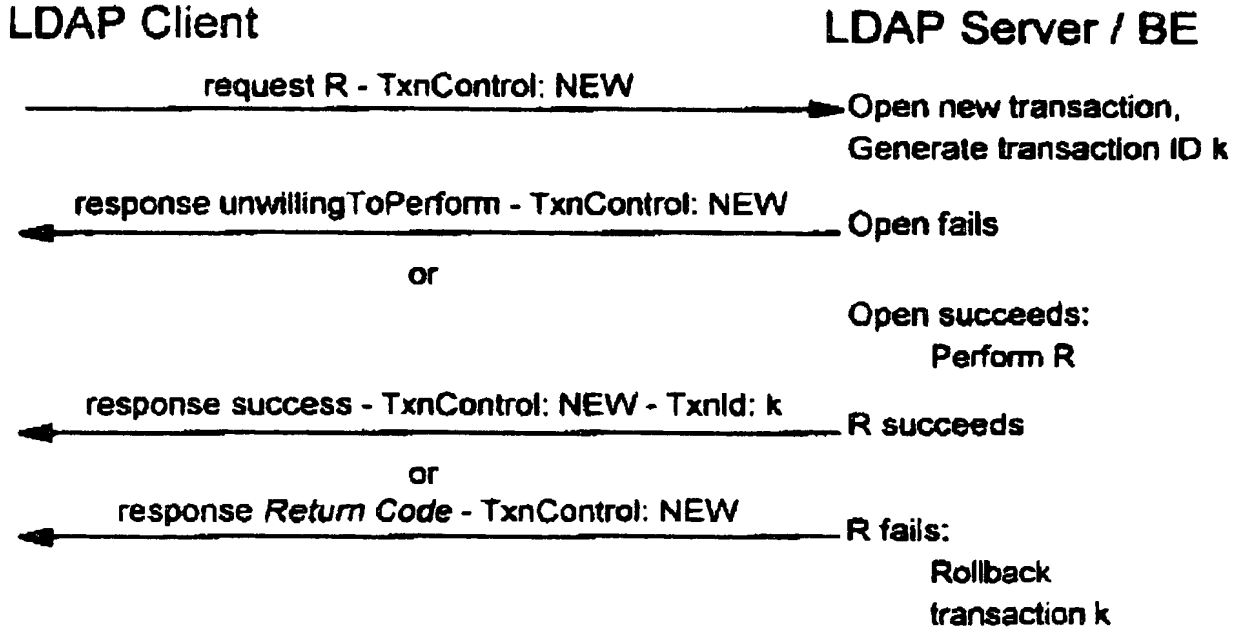

FIG. 3K

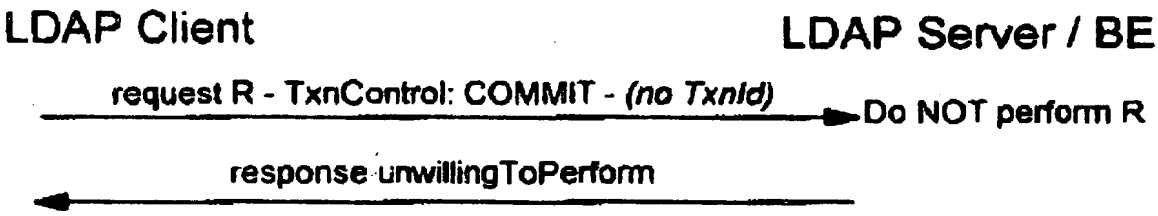

FIG. 3L

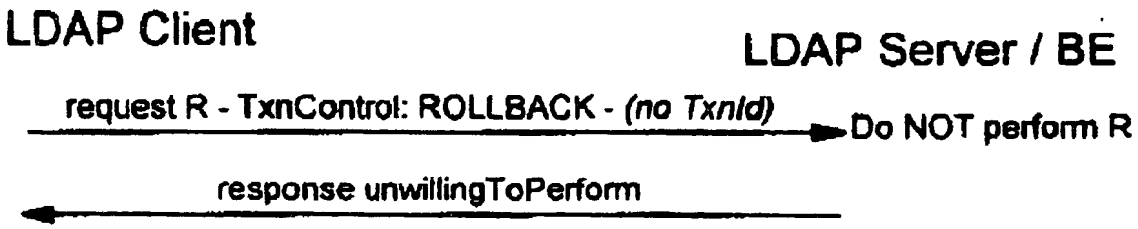

FIG. 3M